nag_opt_lsq_check_deriv (e04yac)

1. Purpose

nag_opt_lsq_check_deriv checks that a user-supplied C function for evaluating a vector of functions and the matrix of their first derivatives produces derivative values which are consistent with the function values calculated.

2. Specification

```
#include <nag.h>
#include <nage04.h>

void nag_opt_lsq_check_deriv(Integer m, Integer n,
            void (*lsqfun)(Integer m, Integer n, double x[], double fvec[],
                           double fjac[], Integer tdj, Nag_Comm *comm),
            double x[], double fvec[], double fjac[], Integer tdj,
            Nag_Comm *comm, NagError *fail)
```

3. Description

The function nag_opt_lsq_deriv (e04gbc) for minimizing a sum of squares of m nonlinear functions (or 'residuals'), $f_i(x_1, x_2, \ldots, x_n)$, for $i = 1, 2, \ldots, m$; $m \geq n$, requires the user to supply a C function to evaluate the f_i and their first derivatives. nag_opt_lsq_check_deriv checks the derivatives calculated by such a user-supplied function. As well as the C function to be checked (**lsqfun**), the user must supply a point $x = (x_1, x_2, \ldots, x_n)^T$ at which the check is to be made.

nag_opt_lsq_check_deriv first calls **lsqfun** to evaluate the $f_i(x)$ and their first derivatives, and uses these to calculate the sum of squares $F(x) = \sum_{i=1}^{m} [f_i(x)]^2$, and its first derivatives $g_j = \left.\frac{\partial f}{\partial x_j}\right|_x$, for $j = 1, 2, \ldots, n$. The components of g along two orthogonal directions (defined by unit vectors p_1 and p_2, say) are then calculated; these will be $g^T p_1$ and $g^T p_2$ respectively. The same components are also estimated by finite differences, giving quantities

$$v_k = \frac{F(x + hp_k) - F(x)}{h}, \quad k = 1, 2$$

where h is a small positive scalar. If the relative difference between v_1 and $g^T p_1$ or between v_2 and $g^T p_2$ is judged too large, an error indicator is set.

4. Parameters

m

n

Input: the number m of residuals, $f_i(x)$, and the number n of variables, x_j.
Constraint: $1 \leq \mathbf{n} \leq \mathbf{m}$.

lsqfun

lsqfun must calculate the vector of values $f_i(x)$ and their first derivatives $\frac{\partial f_i}{\partial x_j}$ at any point x. (The minimization routine nag_opt_lsq_deriv (e04gbc) gives the user the option of resetting a parameter, **comm->flag**, to terminate the minimization process immediately. nag_opt_lsq_check_deriv will also terminate immediately, without finishing the checking process, if the parameter in question is reset to a negative value.)
The specification of **lsqfun** is:

```
void lsqfun(Integer m, Integer n, double x[], double fvec[],
            double fjac[], Integer tdj, Nag_Comm *comm),
```

m

n

Input: the numbers m and n of residuals and variables, respectively.

x[n]

Input: the point x at which the values of the f_i and the $\dfrac{\partial f_i}{\partial x_j}$ are required.

fvec[m]

Output: unless **comm->flag** is reset to a negative number, then **fvec**$[i-1]$ must contain the value of f_i at the point x, for $i = 1, 2, \ldots, m$.

fjac[m*tdj]

Output: unless **comm->flag** is reset to a negative number, then the value in **fjac**$[(i-1)$***tdj**$+j-1]$ must be the first derivative $\dfrac{\partial f_i}{\partial x_j}$ at the point **x**, for $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$.

tdj

Input: the last dimension of the array **fjac** as declared in the function from which nag_opt_lsq_check_deriv is called.

comm

Pointer to structure of type Nag_Comm; the following members are relevant to **lsqfun**.

flag – Integer

Input: **comm->flag** will be set to 2.

Output: if **lsqfun** resets **comm->flag** to some negative number then nag_opt_lsq_check_deriv will terminate immediately with the error indicator **NE_USER_STOP**. If **fail** is supplied to nag_opt_lsq_check_deriv, **fail.errnum** will be set to the user's setting of **comm->flag**.

first – Boolean

Input: will be set to **TRUE** on the first call to **lsqfun** and **FALSE** for all subsequent calls.

nf – Integer

Input: the number of calls made to **lsqfun** including the current one.

user – double *

iuser – Integer *

p – Pointer

The type Pointer will be `void *` with a C compiler that defines `void *` and `char *` otherwise.

Before calling nag_opt_lsq_check_deriv these pointers may be allocated memory by the user and initialised with various quantities for use by **lsqfun** when called from nag_opt_lsq_check_deriv.

The array **x** must **not** be changed within **lsqfun**.

x[n]

Input: **x**$[j-1]$ $(j = 1, 2, \ldots, n)$ must be set to the co-ordinates of a suitable point at which to check the derivatives calculated by **lsqfun**. 'Obvious' settings, such as 0.0 or 1.0, should not be used since, at such particular points, incorrect terms may take correct values (particularly zero), so that errors can go undetected. For a similar reason, it is preferable that no two elements of **x** should have the same value.

fvec[m]

Output: unless **comm->flag** is set negative in the first call of **lsqfun**, **fvec**$[i-1]$ contains the value of f_i at the point given in **x**, for $i = 1, 2, \ldots, m$.

fjac[m][tdj]

Output: unless **comm->flag** is set negative in the first call of **lsqfun**, **fjac**$[i-1][j-1]$ contains the value of the first derivative $\frac{\partial f_i}{\partial x_j}$ at the point given in **x**, as calculated by **lsqfun**, for $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$.

tdj

Input: the second dimension of the array **fjac** as declared in the function from which nag_opt_lsq_check_deriv is called.
Constraint: **tdj** $\geq$ **n**.

comm

Input/Output: structure containing pointers for communication to the user defined function; see the above description of **lsqfun** for details. If the user does not need to make use of this communication feature the null pointer `NAGCOMM_NULL` may be used in the call to nag_opt_lsq_check_deriv; **comm** will then be declared internally for use in calls to **lsqfun**.

fail

The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings

NE_USER_STOP

User requested termination, user flag value = ⟨*value*⟩.

This exit occurs if the user sets **comm->flag** to a negative value in **lsqfun**. If **fail** is supplied the value of **fail.errnum** will be the same as the user's setting of **comm->flag**. The check on **lsqfun** will not have been completed.

NE_INT_ARG_LT

On entry, **n** must not be less than 1: **n** = ⟨*value*⟩.

NE_2_INT_ARG_LT

On entry, **m** = ⟨*value*⟩ while **n** = ⟨*value*⟩. These parameters must satisfy **m** $\geq$ **n**.
On entry, **tdj** = ⟨*value*⟩ while **n** = ⟨*value*⟩. These parameters must satisfy **tdj** $\geq$ **n**.

NE_ALLOC_FAIL

Memory allocation failed.

NE_DERIV_ERRORS

Large errors were found in the derivatives of the objective function.

The user should check carefully the derivation and programming of expressions for the $\frac{\partial f_i}{\partial x_j}$, because it is very unlikely that **lsqfun** is calculating them correctly.

6. Further Comments

nag_opt_lsq_check_deriv calls **lsqfun** three times.

Before using nag_opt_lsq_check_deriv to check the calculation of the first derivatives, the user should be confident that **lsqfun** is calculating the residuals correctly.

6.1. Accuracy

fail.code is set to **NE_DERIV_ERRORS** if

$$(v_k - g^T p_k)^2 \geq h \times ((g^T p_k)^2 + 1)$$

for $k = 1$ or 2. (See Section 3 for definitions of the quantities involved.) The scalar h is set equal to $\sqrt{\epsilon}$, where ϵ is the ***machine precision*** as given by nag_machine_precision (X02AJC).

7. See Also

nag_opt_lsq_deriv (e04gbc)

8. Example

Suppose that it is intended to use nag_opt_lsq_deriv (e04gbc) to find least-squares estimates of x_1, x_2 and x_3 in the model

$$y = x_1 + \frac{t_1}{x_2 t_2 + x_3 t_3}$$

using the 15 sets of data given in the following table:

y	t_1	t_2	t_3
0.14	1.0	15.0	1.0
0.18	2.0	14.0	2.0
0.22	3.0	13.0	3.0
0.25	4.0	12.0	4.0
0.29	5.0	11.0	5.0
0.32	6.0	10.0	6.0
0.35	7.0	9.0	7.0
0.39	8.0	8.0	8.0
0.37	9.0	7.0	7.0
0.58	10.0	6.0	6.0
0.73	11.0	5.0	5.0
0.96	12.0	4.0	4.0
1.34	13.0	3.0	3.0
2.10	14.0	2.0	2.0
4.39	15.0	1.0	1.0

The following program could be used to check the first derivatives calculated by the required function **lsqfun**. (The tests of whether **comm->flag** $\neq$ 0 or 1 in **lsqfun** are present for when **lsqfun** is called by nag_opt_lsq_deriv (e04gbc). nag_opt_lsq_check_deriv will always call **lsqfun** with **comm->flag** set to 2.)

8.1. Program Text

```
/* nag_opt_lsq_check_deriv (e04yac) Example Program
 *
 * Copyright 1991 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nage04.h>

#ifdef NAG_PROTO
static void lsqfun(Integer m, Integer n, double x[], double fvec[],
                   double fjac[], Integer tdj, Nag_Comm *comm);
#else
static void lsqfun();
#endif

main()
{
#define MMAX 15
#define NMAX 3
#define Y(I) comm.user[I]
#define T(I,J) comm.user[(I)*NMAX + (J) + MMAX]

  double fjac[MMAX][NMAX], fvec[MMAX], x[NMAX];
  double work[MMAX + MMAX*NMAX];
  Integer i, j, m, n, tdj;
  Nag_Comm comm;
  static NagError fail;
```

```
  Vprintf("e04yac Example Program Results\n");
  Vscanf(" %*[^\n]"); /* Skip heading in data file */

  n = 3;
  m = 15;
  tdj = NMAX;

  fail.print = TRUE;

  /* Allocate memory to communication array */
  comm.user = work;

  /* Observations t (j = 0, 1, 2) are held in T(i, j)
   * (i = 0, 1, 2, . . . , 14) */
  for (i = 0; i < m; ++i)
    {
      Vscanf("%lf", &Y(i));
      for (j = 0; j < n; ++j) Vscanf("%lf", &T(i,j));
    }

  /* Set up an arbitrary point at which to check the 1st derivatives */
  x[0] = 0.19;
  x[1] = -1.34;
  x[2] = 0.88;
  Vprintf("\nThe test point is ");
  for (j = 0; j < n; ++j)
    Vprintf(" %9.3e", x[j]);
  Vprintf("\n");

  fail.print = TRUE;
  e04yac(m, n, lsqfun, x, fvec, (double *)fjac, tdj, &comm, &fail);

  if (fail.code != NE_NOERROR) exit(EXIT_FAILURE);

  Vprintf("\nDerivatives are consistent with residual values.\n");
  Vprintf("\nAt the test point, lsqfun() gives\n\n");
  Vprintf("      Residuals                   1st derivatives\n");
  for (i = 0; i < m; ++i)
    {
      Vprintf("     %9.3e  ", fvec[i]);
      for (j = 0; j < n; ++j)
        Vprintf("     %9.3e", fjac[i][j]);
      Vprintf("\n");
    }
  exit(EXIT_SUCCESS);
}

#ifdef NAG_PROTO
static void lsqfun(Integer m, Integer n, double x[], double fvec[],
                   double fjac[], Integer tdj, Nag_Comm *comm)
#else
     static void lsqfun(m, n, x, fvec, fjac, tdj, comm)
     Integer m, n;
     double x[], fvec[], fjac[];
     Integer tdj;
     Nag_Comm *comm;
#endif
{
  /* Function to evaluate the residuals and their 1st derivatives. */

#define YC(I) comm->user[(I)]
#define TC(I,J) comm->user[(I)*NMAX + (J) + MMAX]
#define FJAC(I,J) fjac[(I)*tdj + (J)]

  Integer i;
  double denom, dummy;

  for (i = 0; i < m; ++i)
    {
      denom = x[1]*TC(i,1) + x[2]*TC(i,2);
```

```
        if (comm->flag != 1)
          fvec[i] = x[0] + TC(i,0)/denom - YC(i);
        if (comm->flag != 0)
          {
            FJAC(i,0) = 1.0;
            dummy = -1.0 / (denom * denom);
            FJAC(i,1) = TC(i,0)*TC(i,1)*dummy;
            FJAC(i,2) = TC(i,0)*TC(i,2)*dummy;
          }
      }
  }                               /* lsqfun */
```

8.2. Program Data

```
e04yac Example Program Data

 0.14  1.0 15.0  1.0
 0.18  2.0 14.0  2.0
 0.22  3.0 13.0  3.0
 0.25  4.0 12.0  4.0
 0.29  5.0 11.0  5.0
 0.32  6.0 10.0  6.0
 0.35  7.0  9.0  7.0
 0.39  8.0  8.0  8.0
 0.37  9.0  7.0  7.0
 0.58 10.0  6.0  6.0
 0.73 11.0  5.0  5.0
 0.96 12.0  4.0  4.0
 1.34 13.0  3.0  3.0
 2.10 14.0  2.0  2.0
 4.39 15.0  1.0  1.0
```

8.3. Program Results

```
e04yac Example Program Results

The test point is  1.900e-01 -1.340e+00 8.800e-01

Derivatives are consistent with residual values.

At the test point, lsqfun() gives

      Residuals                  1st derivatives
     -2.029e-03       1.000e+00     -4.061e-02     -2.707e-03
     -1.076e-01       1.000e+00     -9.689e-02     -1.384e-02
     -2.330e-01       1.000e+00     -1.785e-01     -4.120e-02
     -3.785e-01       1.000e+00     -3.043e-01     -1.014e-01
     -5.836e-01       1.000e+00     -5.144e-01     -2.338e-01
     -8.689e-01       1.000e+00     -9.100e-01     -5.460e-01
     -1.346e+00       1.000e+00     -1.810e+00     -1.408e+00
     -2.374e+00       1.000e+00     -4.726e+00     -4.726e+00
     -2.975e+00       1.000e+00     -6.076e+00     -6.076e+00
     -4.013e+00       1.000e+00     -7.876e+00     -7.876e+00
     -5.323e+00       1.000e+00     -1.040e+01     -1.040e+01
     -7.292e+00       1.000e+00     -1.418e+01     -1.418e+01
     -1.057e+01       1.000e+00     -2.048e+01     -2.048e+01
     -1.713e+01       1.000e+00     -3.308e+01     -3.308e+01
     -3.681e+01       1.000e+00     -7.089e+01     -7.089e+01
```